VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Implementace síťové varianty hry Srdce pro Android

Implementation of Network Version of Heartz for Android

2014 Michal Zlacký

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Zlacký**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Implementace síťové varianty hry Srdce pro Android
Implementation of Network Version of Heartz for Android

Zásady pro vypracování:

Cílem bakalářské práce je implementace síťové varianty hry Srdce pro Android. Hra umožní založit novou hru nebo se připojit k již založené hře případně zkusit štěstí na veřejném serveru. Hra bude využívat pro komunikaci rozhraní TCP/IP a Bluetooth.

1. Proveďte rešerši vývojových prostředí pro vývoj aplikací na platformě Android.
2. Nastudujte možnosti platformy Android týkající se síťové komunikace.
3. Implementujte síťovou variantu hry Srdce.
4. Otestujte výslednou aplikaci a proveďte zátěžové testy jednotlivých síťových rozhraní.

Seznam doporučené odborné literatury:

ZECHNER, Mario. Beginning Android games. New York: Apress, c2011, xvi, 669 s. ISBN 978-1-4302-3042-7

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Roman Meca**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne 7.5.2014

.....
podpis studenta

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, zvláště mému vedoucímu práce, protože bez nich by tato práce nevznikla.

Abstrakt

Cílem práce je implementace síťové varianty hry Srdce určené pro operační systém Android. V teoretické části je stručně popsán samotný operační systém a jeho možnosti v oblasti síťové komunikace. Praktická část se zabývá samotnou implementací aplikace a testováním vytvořené knihovny tříd.

Klíčová slova: Android, třída, balíček

Abstract

The aim of this bachelor thesis is the implemetation of an online variation of game Hearts designed for operation system Android. In the theoretical part the operating system itself is decribed as well as its functions in the area of network communication. The practical part deals with the applications implementation and testing of the created class library.

Keywords: Android, class, package

Seznam použitých zkratek a symbolů

IDE	–	Integrated Development Environment
SDK	–	Software Development Kit
NDK	–	Native Development Kit
Wi-Fi	–	Wireless Fidelity
API	–	Application Interface
RFCOMM	–	Radio Frequency Communication
UUID	–	Universally unique identifier
CDMA	–	Code Division Multiple Access
VPN	–	Virtual private network
JIT	–	Just in time
NFC	–	Near Field Communication
ADT	–	Android Development Tools
URI	–	Uniform Resource Identifier
HTTP	–	Hypertext Transfer Protocol
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Chytré telefony a tablety se staly nedílnou součástí dnešního života. Denně vzniká spousta nových aplikací, které nám ulehčují práci nebo naopak her, u kterých si odpočineme. Mobilní platformy se staly cílovým segmentem pro mnoho vývojářských firem.

V této době je také téměř nutnost být neustále online neboli připojen k internetu. Téměř každý si potřebuje někdy zkontrolovat svou e-mailovou schránku, vyhledat vlakové či autobusové spojení nebo změnit status na Facebooku. To lze s využitím mobilního telefonu učinit téměř kdekoli.

Na poli operačních systémů pro mobilní zařízení má největší podíl systém Android. Důvodů proč si tento systém získal takovou oblibu je několik. V prvé řadě je to jednoduchost, se kterou lze pro tento systém vyvíjet aplikace. Díky možnosti psát kód v jazyce Java je přechod na tuto platformu otázkou několika měsíců. Dalším z důvodů je integrace vývojových nástrojů do známého prostředí Eclipse IDE. Společnost Google navíc vyvíjí vlastní prostředí s názvem Android Studio určené výhradně pro tento systém.

Tento systém se také neustále vyvíjí. Za šest let své existence prošel řadou změn, aby si udržel své výsadní postavení. Aktuálně se nachází ve verzi 4.4 také známou pod kódovým označením KitKat.

Z hlediska síťové komunikace podporuje systém Android veškeré moderní technologie. Mobilní operátoři nabízejí stále výhodnější datové tarify a díky tomu stoupá obliba přenosu dat pomocí mobilního IP protokolu. Další možností je připojit se prostřednictvím bezdrátových sítí Wi-Fi. Pro přenos dat mezi zařízeními na krátkou vzdálenost lze využít technologii BlueTooth nebo nejnověji také technologii Wi-Fi Direct, která nabízí vyšší přenosové rychlosti a také větší dosah. V novějších zařízeních lze nastavit sdílení připojení nebo práci v režimu přístupového bodu, i když tento režim je u některých operátorů zablokován.

U všech těchto technologií můžeme zvolit, na jaké vrstvě budeme komunikovat. Android nabízí možnost jednoduše komunikovat prostřednictvím HTTP protokolu nebo lze pro úplnou kontrolu nad přenosem zvolit paketovou komunikaci přes TCP nebo UDP.

Tato bakalářská práce se zabývá možnostmi síťové komunikace v systému Android. Jako názorná ukázka byla vytvořena knihovna tříd starající se o komunikaci v aplikacích typu Server—Klient. Tato knihovna je důkladně popsána ve třetí čtvrté kapitole. S pomocí této knihovny byla vytvořena aplikace *HOT Hearts*, což je síťová varianta karetní hry Srdce pro čtyři hráče.

2 Android a jeho vývojová prostředí

2.1 Systém Android

2.1.1 Historie firmy

Firma Android Inc. byla založena v Kalifornii, v říjnu roku 2003. Jejími spoluzakladateli byli Andy Rubin, Rich Miner, Nick Sears a Chris White. Prvotním cílem firmy bylo vyvinout vyspělý operační systém pro digitální fotoaparáty, od této myšlenky však bylo upuštěno a firma se přeorientovala na vývoj vlastního operačního systému.

Tento krok neunikl společnosti Google, která v roce 2005 odkoupila firmu Android a tím rozpoutala spekulace, že se chystá vstoupit na trh s mobilními telefony. První verze SDK byla ohlášena 5.listopadu 2007 [2]. O týden později byla tato verze dostupná vývojářům. Prvním telefonem s operačním systémem Android byl T-Mobile G1. Ten byl představen v září 2008. O měsíc později bylo SDK Androidu uvolněno jako open-source, což se ukázalo jako katalyzátor masivního rozmachu zařízení běžících na tomto systému. Dnes se odhaduje počet denně aktivovaných zařízení na více než jeden milion [3].

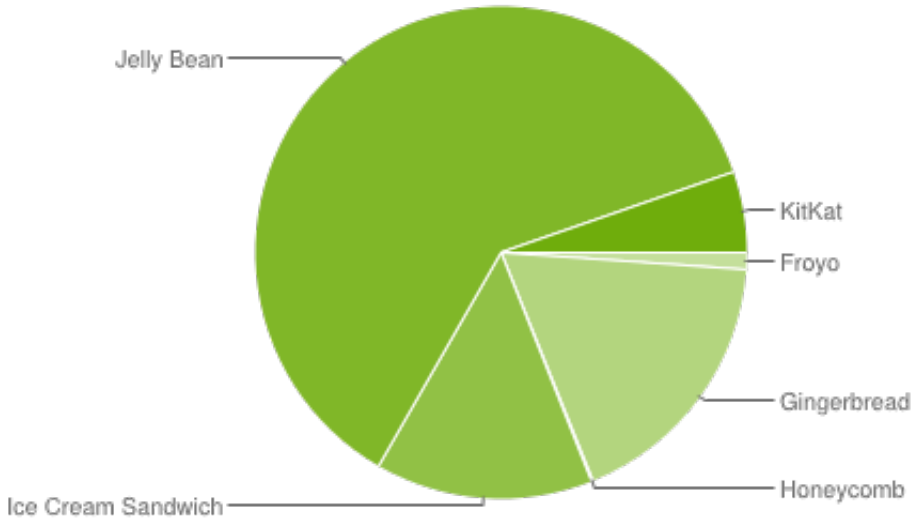
2.1.2 Jednotlivé verze systému

Operační systém Android je založen na Linuxovém jádru v.2.6 a 3.x a je volně dostupný ke komerčnímu i nekomerčnímu využití. Oficiální verze jsou vydávány přímo firmou Google a většinou přináší novou funkcionalitu ve formě API nebo nových vývojových nástrojů.

- 1.0 - první verze, objevila se v zařízení T-Mobile G1
- 1.1 - oprava chyb předchozí verze.
- 1.5 (Cupcake) - přidána podpora vkládání nativních knihoven do Android aplikací. Do té doby byla nutnost psát knihovny přímo v jazyce Java. Z uživatelských funkcí to byla nová softwarová klávesnice, funkce nahrávání a přehrávání videa nebo vylepšení funkce copy-paste.
- 1.6 (Donut) - přídáná podpora CDMA a VPN, podpora rozličných velikostí obrazovky, vylepšen Android Market, kamera, galerie.
- 2.0 - 2.1 (Éclair) - možnost více účtů na jednom zařízení, navidgace Google Mapy, funkce Speech-to-Text, Nexus One - první telefon se systémem bez modifikací, přidání podpora více-dotykových zařízení.
- 2.2 (Froyo) - přidána podpora Wi-fi Hotspot, možnost jak udělat ze svého telefonu mobilní přístupový bod, přidán JIT kompilátor do Dalvik VM, software na kterém běží všechny java aplikace na systému android. V určitých situacích tento kompilátor urychluje spouštění android aplikací
- 2.3 (Gingerbread) - podpora přední kamery, NFC, lepší správa baterie, přidán nový souběžný garbage collector do Dalvik VM.

- 3.0 (Honeycomb) - verze android pro tablet zařízení. Tato verze přidala spoustu rozšíření na podporu vysokého rozlišení, připojení USB zařízení jako klávesnic, myší a joysticků. Tato verze byla určena pouze pro tablet zařízení, vylepšen multitasking.
- 4.0.1 - 4.0.4 (Ice Cream Sandwich) - sjednocená verze pro telefony a tablety, technologie Android Beam, Wi-Fi Direct, BlueTooth HDP, ukazatel spotřeby dat.
- 4.1 - 4.3 (Jelly Bean) - aktuálně nejrozšířenější verze, vylepšen způsob kompozice a renderace samotného uživatelského rozhraní. Přidána podpora OpenGLEs 3.0.
- 4.4 (Kit Kat) - zatím poslední verze, technologie HCE - aplikace mohou emulovat platební karty, přidán tiskový framework - tisk přes Wi-fi, krokoměr a detektor chůze, nahrávání obrazovky, technologie HTTP Live Streaming, nové BlueTooth profily.

Zastoupení jednotlivých verzí je zobrazeno na obrázku č.1.



Obrázek 1: Zastoupení verzí systému Android ke dni 1.4.2014 [4]

2.1.3 Software Development Kit

Android SDK je nezbytnou součástí pro vývoj v operačním systému Android. Skládá se z jednotlivých API knihoven a vývojových nástrojů nezbytných pro kompilaci, testování a ladění výsledných aplikací. Součástí Development Kitu je:

- SDK tools - tento balíček obsahuje nástroje pro ladění a testování a další utility, které jsou potřebné pro vývoj.
- SDK Platform - toto rozšíření existuje pro každou verzi systému.

- SDK Platform-tools - obsahuje specifické nástroje
- Documentation - kopie nejnovější dokumentace.
- Sources for Android SDK - kopie zdrojových kódů, užitečných pro ladění aplikací
- Samples for SDK - kolekce vzorových příkladů, které ukazují možnosti vývojového prostředí.
- Google APIs - balíček knihoven Google
- Android Support - statické knihovny, díky které je možno použít API vyšších verzí ve starších prostředích.
- Google Play Billing - statická knihovna, umožňující integraci platebních služeb prostřednictvím Google Play
- Google Play Licensing - tento balíček umožňuje provádět verifikaci licencí.

Tento výčet lze rozšířit také o knihovny třetích stran.

2.2 Vývojová prostředí

Pro vývoj aplikací pro systém Android stačí mít nainstalováno SDK a nějaký textový editor, ale tato možnost je složitá a neefektivní. Společnost Google nabízí dvě alternativy, které výrazně ulehčují vývoj a testování pro tento systém.

2.2.1 Eclipse + ADT Plugin

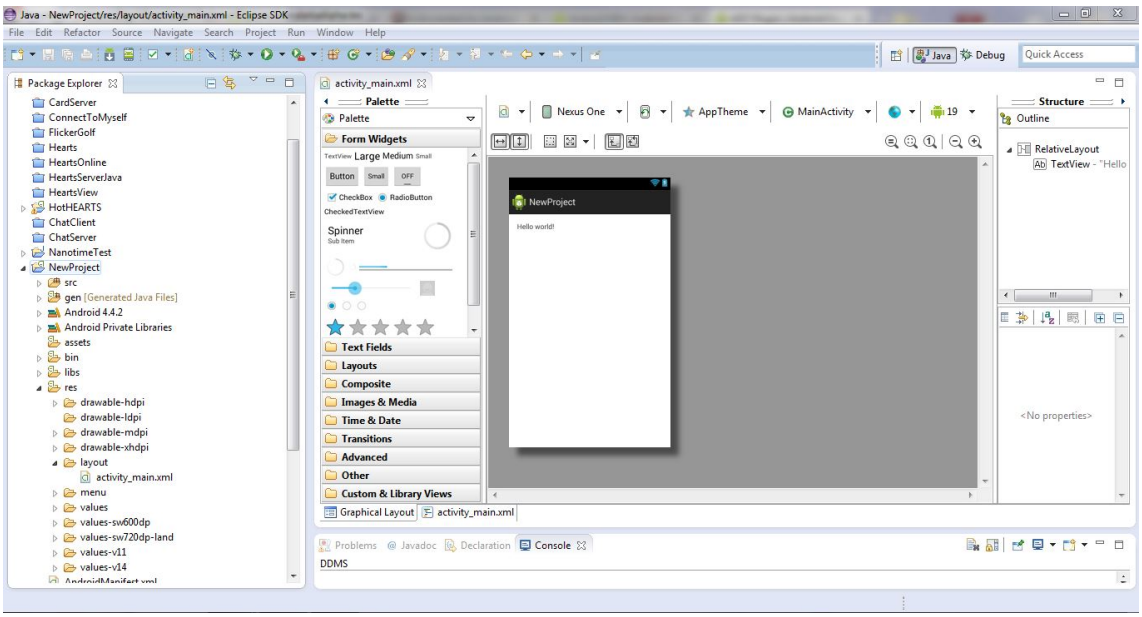
Eclipse je open-source IDE, které lze využít k vývoji Java aplikací. Aby bylo možné vyvíjet také mobilní aplikace, lze k tomuto IDE doinstalovat ADT. Tento doplněk rozšiřuje možnosti Eclipse a tvorbu nových projektů přímo pro systém Android, umožňuje vytvářet grafická uživatelská rozhraní, ladit aplikace pomocí SDK nástrojů a v neposlední řadě také přidává možnost exportovat podepsané aplikace.

Společnost Google nabízí verzi Eclipse, ve které je tento doplněk již obsažen. Na obrázku č.2 je zobrazena tvorba uživatelského rozhraní v tomto vývojovém prostředí.

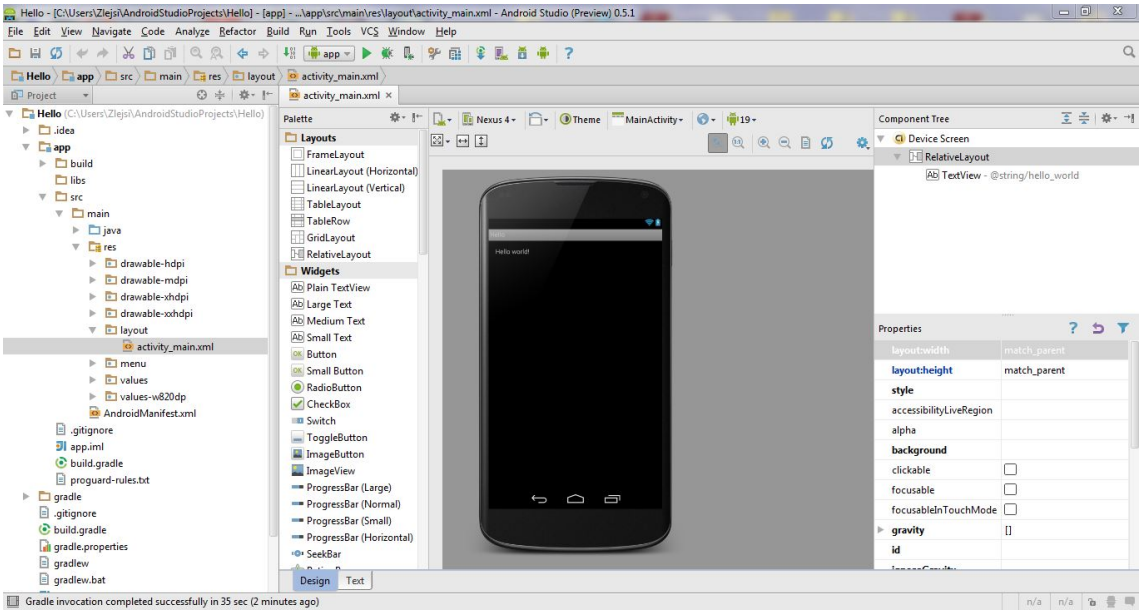
2.2.2 Android Studio

Android Studio je nové vývojové prostředí založené na IntelliJ IDEA. Stejně jako Eclipse s nainstalovaným ADT nabízí integrované nástroje pro vývoj, testování a ladění aplikací. V době psaní této bakalářské práce se toto prostředí nachází ve verzi Early Access Preview, to znamená, že některé funkce nemusí být kompletní nebo mohou chybět úplně. Do budoucna se však očekává, že bude toto prostředí preferováno před Eclipse IDE.

Na obrázku č.3 lze vidět výchozí nastavení Android Studia při tvorbě nového projektu.



Obrázek 2: Prostředí Eclipse IDE



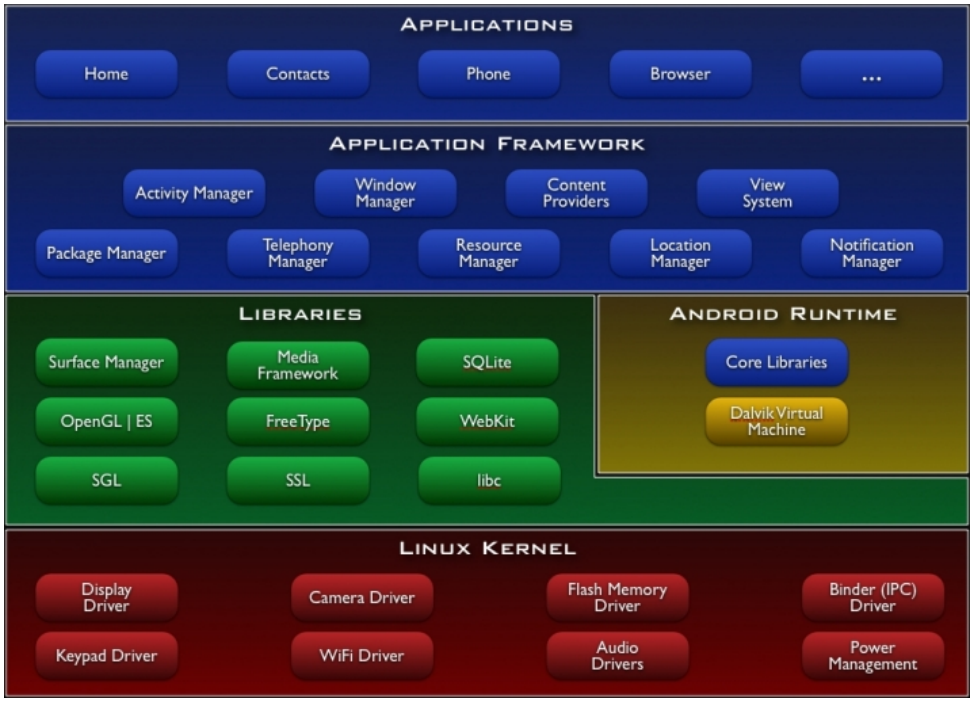
Obrázek 3: Prostředí Android Studio

2.3 Architektura systému

Architektura systému Android se skládá z pěti vrstev.

- Linuxové jádro - toto je nejnižší vrstva systému. Má na starosti spojení mezí hardwarem a softwarovými prostředky. Taktéž zajišťuje bezpečnost. správu paměti, síťové prostředky a ovladače.
- Knihovny - v této vrstvě se nachází knihovny napsané v nativním jazyce C nebo C++. Tyto knihovny nejsou přístupny pro vývojáře.
- Android Runtime - zde se nachází virtuální stroj Dalvik. Tento stroj má na starost překlad aplikací z Java byte kódu do Dalvik formátu. Na této úrovni se také nachází knihovny převzaté z jazyka Java.
- Aplikační framework - tato vrstva je z vývojářského hlediska nedůležitější. Obsahuje správce jednotlivých komponent systému.
- Aplikace - na nejvyšší vrstvě se nachází veškeré aplikace, které byly napsány v jazyce Java. V základu je zde několik předinstalovaných aplikací.

Celková architektura je vyobrazena na obrázku č.4.



Obrázek 4: Architektura systému Android

2.4 Aplikace

2.4.1 Komponenty

Mezi základní komponenty patří tyto:

- Aktivity - je základním prvkem každé aplikace. Nalézají se na ní prvky uživatelského rozhraní. Jejím prostřednictvím je prováděna interakce s uživatelem.
- ContentProvider - dovoluje aplikacím pracovat s obsahem telefonu.
- Service - služby, jež dokáží běžet na pozadí nezávisle na spuštěné aktivitě.
- BroadcastRecevier - notifikace informující uživatele o proběhlé události.

2.4.2 Android manifest

Tento XML soubor je součástí každé aplikace. Obsahuje definici aplikace, všech jejích komponent, ale především také výčet práv, která tato aplikace obsahuje.

2.4.3 Práva

Aby bylo možno přistupovat k některým službám systému, je nutné definovat práva. Tyto práva jsou zapsána v souboru manifestu. V případě neuvedení je vyvolána výjimka typu SecurityException. Ukázku manifestu včetně deklarace práv je možno vidět na výpisu č.1.

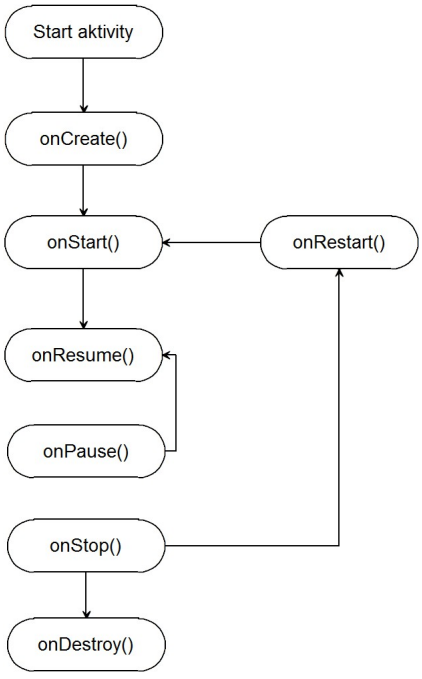
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.zlejsi.
    cardgui" android:versionCode="1" android:versionName="1.0">
<uses-sdk android:minSdkVersion="9" android:targetSdkVersion="19"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"/>
<application android:name="com.zlejsi.cardgui.HeartsAP" android:allowBackup="true" android:icon
    ="@drawable/ic_launcher" android:label="@string/app_name" android:theme="@style/
    AppTheme">
<activity  android:name="com.zlejsi.cardgui.CardGUIActivity" android:label="@string/app_name"
    android:configChanges="keyboard|keyboardHidden|orientation">
<intent- filter >
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent- filter >
</ activity >
</application>
</manifest>
```

Výpis 1: Ukázka manifestu

2.4.4 Životní cyklus aktivity

Každá aktivita obsahuje metody, které jsou volány v určité fázi jejího vzniku, případně zániku. Mezi tyto metody patří:

- onCreate - volá se při inicializaci aktivity. Nastavuje se zde například uživatelské rozhraní.
- onStart - tato metoda se volá, jakmile je aktivita viditelná pro uživatele.
- onRestart - tato metoda je zavolána, když uživatel znovu spustí aktivitu, kterou zastavil.
- onResume - metoda je volána při návratu z pozadí do popředí.
- onPause - tato metoda se volá, jakmile se aktivita dostane do pozadí. V tomto bodě může aktivitu ukončit GC.
- onStop - je volána např. při stisku tlačítka Zpět, tj. uživatel opustil aktivitu.
- onDestroy - volá se při ukončení aktivity.



Obrázek 5: Životní cyklus aktivity

3 Možnosti platformy Android v oblasti síťové komunikace

3.1 Balíčky

SDK systému android obsahuje několik balíčků, ve kterých se nalézají třídy potřebné pro síťovou komunikaci. Tyto třídy umožňují vytvářet spojení prostřednictvím mobilních sítí, sítí Wi-Fi nebo BlueTooth. Jednotlivé balíčky jsou zobrazeny v tabulce č.1.

Balíček	*Popis*
java.net	Poskytuje síťově zaměřené třídy, zahrnuje TCP a UDP sokety, Internetové protokoly a obecné HTTP. Je stejný jako v klasické Javě.
java.io	Tento balíček nesouvisí přímo se síťovou komunikací, poskytuje však důležité třídy, které jsou používány v soketové komunikaci
java.nio	Obsahuje třídy reprezentující buffery a specifické datové typy
org.apache.*	V těchto balíčcích jsou obsaženy funkce pro HTTP komunikaci
android.net	Tento balíček obsahuje URI třídu, která se často používá v Android aplikacích
android.net.http	Manipulace s SSL certifikáty
android.net.wifi	Třídy pro správu všech aspektů Wifi komunikace na Android platformě
android.bluetooth	Třídy pro komunikaci prostřednictvím BlueTooth.

Tabulka 1: Tabulka balíčků

3.2 Technologie

3.2.1 Wi-Fi

Třídy pro správu Wi-Fi jsou obsaženy v balíčku android.net.wifi. Zde nalezneme třídu WifiManager, která se stará o veškeré aspekty spojené s touto technologií. Pro práci s touto třídou je zapotřebí vytvořit její instanci voláním metody getSystemService třídy Context. Následně jsou k dispozici metody umožňující zjišťovat stav spojení, počet připojených klientů nebo aktualizovat popis již existující sítě.

3.2.2 Wi-Fi Hotspot

Od verze 2.2 se mohou mobilní zařízení chovat jako přístupový bod. V tomto režimu je zařízení odpojeno od sítě Wi-Fi a sdílí mobilní připojení přes softwarový *access point*. V současnosti není v SDK obsaženo API, sloužící ke správě této technologie.

3.2.3 Wi-Fi Direct

Technologie Wi-Fi Direct je nové označení, schválené organizací Wi-Fi Alliance. Umožňuje přenos dat mezi různými zařízeními bez nutnosti vytváření přístupového bodu. Oproti technologii BlueTooth nabízí větší rychlost, dosah a lepší zabezpečení přenosu [6].

3.2.4 Bluetooth

Ke správě Bluetooth připojení slouží Bluetooth API. Toto API, které se nalézá v balíčku android.bluetooth, umožňuje aplikacím bezdrátové spojení s dalšími Bluetooth zařízeními. Dále lze provádět následující operace:

- Vyhledávat Bluetooth zařízení
- Vytvářet RFCOMM kanály
- Připojit se k dalším zařízením skrze vyhledávací službu
- Přenášet data mezi zařízeními

V manifestu aplikace je nutné definovat právo android.permission.BLUETOOTH. Navíc některé služby jako je vyhledávání zařízení vyžadují také právo android.permission.BLUETOOTH_ADMIN.

3.3 Specifická omezení

V systému Android existují určitá omezení vztahující se na síťovou komunikaci. Při spuštění déle trvající operace, jakou je například stahování souboru, může dojít k tomu, že aplikace přestane reagovat. Z tohoto důvodu je od verze 3.0 nutné provádět veškeré sítové operace v separátním vlákně. Pří pokusu spustit operaci v hlavním vlákně je vyvolána výjimka typu NetworkingOnMainThreadException. Pokud potřebujeme tuto ochranu obejít, je nutno nastavit v manifestu cílovou verzi nižší než 10 nebo pomocí třídy ThreadPolicy zmírnit tato omezení.

```
ThreadPolicy tp = ThreadPolicy.LAX;
StrictMode.setThreadPolicy(tp);
```

Vzhledem k rozšíření počtu zařízení pracujících na verzi 4.0 a vyšší je již první možnost nevhodná.

3.4 HTTP nebo Soket?

Android poskytuje několik možností práce v síti. Jednou z možností je využít HTTP protokol. K jeho správě slouží v třídy HttpURLConnection nebo HttpClient. Další možnost, a to je ta ,kterou preferuji v této práci, je pracovat na transportní vrstvě nad protokoly TCP nebo UDP. Pro tento případ je v SDK obsažena třída Socket, známá již z jazyka Java.

3.4.1 HTTP

Práce s protokolem HTTP je snadná, hlavně díky třídě HttpURLConnection. Způsob jejího použití lze vyjádřit v následujících krocích:

1. Získání nové instance voláním URL.openConnection() a následným přetypováním na HttpURLConnection.
2. Příprava dotazu Základní částí dotazu je jeho URI (Universal Resource Interface). Hlavička dotazu také může obsahovat metadata.
3. Nahrání těla dotazu. Instance musí být konfigurovány metodou setDoOutput(**true**). Samotný přenos je zapsán do proudu, vráceného metodou getOutputStream().
4. Přečtení odpovědi. Hlavička odpovědi typicky obsahuje metadata, jakými jsou typ a délka obsahu, datum poslední modifikace atd. Získání samotného obsahu lze provést vytvořením vstupního proudu pomocí metody getInputStream() a jeho následným přečtením. Pokud odpověď žádná data neobsahuje, je vrácen prázdný proud.
5. Odpojení. Jakmile je odpověď přečtena, instance HttpURLConnection by měla být uzavřena voláním metody disconnect().

Příklad práce s třídou HttpURLConnection je uveden na výpisu č.2.

```
URL url = new URL("http://www.vsb.cz");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
try
{
    InputStream in = new BufferedInputStream(urlConnection.getInputStream());
     readStream(in);
}
catch(IOException e){}
 finally
{
    urlConnection.disconnect();
}
```

Výpis 2: Třída HttpURLConnection

3.4.2 UDP

Základní třídou pro práci v protokolu UDP je DatagramSocket. Jejím prostřednictvím se zasílají objekty typu DatagramPacket. Komunikaci můžeme rozdělit do dvou částí:

1. Příjemce:

   Na straně příjemce je nejdřív vytvořeno pole typu byte o velikosti předpokládané datové složky. Toto pole se následně předá instanci DatagramPaketu. Následně je zavolána metoda receive(), která, není-li nastaven parametr setTimeout(), vyčkává na příjem paketu. Jakmile je paket obdržen, naplní se pole bytů obsahem jeho datové části.

2. Odesílatel:

   Odesílatel vytvoří soket a inicializuje jej na žádanou IP adresu. Následně vytvoří pole typu byte a naplní jej daty. Toto pole je předáno instanci DatagramPacketu. Paket je odeslán zavoláním metody send(). O jeho doručení se již dále nestará.

Jednoduchý příklad komunikace je uveden na výpisu č.3

```
SERVER:

String messageStr="Hello Android!";
int server_port = 12345;
DatagramSocket s = new DatagramSocket();
InetAddress local = InetAddress.getByName("192.168.1.102");
int msg_length=messageStr.length();
byte[] message = messageStr.getBytes();
DatagramPacket p = new DatagramPacket(message, msg_length,local,server_port);
s.send(p);

KLIENT:

String text;
int server_port = 12345;
byte[] message = new byte[1500];
DatagramPacket p = new DatagramPacket(message, message.length);
DatagramSocket s = new DatagramSocket(server_port);
s.receive(p);
text = new String(message, 0, p.getLength());
Log.d("Udp tutorial","message:" + text);
s.close();
```

Výpis 3: Komunikace prostřednictvím UDP

V případě zasílání broadcastových zpráv je nutné u odesílatele nastavit příznak setBroadcast(). Příjemce může zprávy přijímat pouze instancí třídy MulitcastSocket. Systém Android v běžném nastavení nepřijímá zprávy neurčené přímo na konkrétní IP adresu. Toto chování lze změnit nastavením zámku. Třídou, která se o tento zámek stará je třída MulticastLock. Nejprve je nutné vytvořit objekt zámku pomocí metody createMulticastLock(), která se nachází ve třídě WifiManager. Následně je potřeba zámek získat, a to prostřednictvím metody aquire(). Po skončení příjmu lze zámek uvolnit metodou release().

3.4.3 TCP

Ke komunikaci prostřednictvím protokolu TCP slouží třída Socket. Na serverové straně je nejprve nutné vytvořit instanci třídy ServerSocket a přiřadit jí port, na kterém bude server naslouchat. To lze provést v konstruktoru nebo voláním metody bind(), které jako parametr předáme IP adresu a port. Samotné naslouchání lze provést zavoláním metody accept(). Tato metoda čeká na spojení ze strany klienta. Jakmile dojde ke spojení, je na serverové straně vytvořen soket, prostřednictvím kterého probíhá komunikace.

Klient vytváří instanci třídy Socket a inicializuje ji na serverovou IP adresu a port, na kterém chce komunikovat. Pokud server naslouchá, je provedeno spojení, v opačném případě je volána výjimka typu SocketException, která identifikuje příčinu neúspěchu.

Jednoduchý příklad vytvoření serveru a klienta je uveden na výpisu níže. Takto vytvořený server příjme pouze jednoho klienta a zbytek ignoruje.

```
SERVER:

ServerSocket sSocket = new ServerSocket(5555);
Socket socket = sSocket.accept();
 PrintWriter  pw = new PrintWriter(socket.getOutputStream(),true);
pw.println("Welcome");


KLIENT:
Socket socket = new Socket("127.0.0.1",5555);
BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
System.out.println(br.readLine());
```

Výpis 4: Komunikace prostřednictvím TCP

Pro vytvoření serveru, jež umí přijímat více klientů, je zapotřebí spouštět volání metody accept() ve smyčce. Navíc je vhodné pracovat s každým klientem ve vlastním vlákně. Příklad vícevláknového serveru je uveden níže.

```
ServerSocket sSocket = new ServerSocket(5555);
while(true)
{
   Socket socket = sSocket.accept();

   ClientThread ct = new ClientThread(socket);
   ct.start();
}
```

V tomto příkladu je každý klient zpracováván ve vlastní třídě, která dědí ze třídy Thread. Do její instance je předán soket a následně je volána metoda start(), která spustí nové vlákno.

Takovýto postup byl použit i při tvorbě aplikace v této bakalářské práci.

3.4.4 RFCOMM

Tento protokol je využíván u komunikace prostřednictvím Bluetooth. Aby mohl být uskutečněn přenos mezi dvěma zařízeními, musí se jedno ze zařízení chovat jako server a druhé jako klient. Zařízení jsou považována za spojená, pokud sdílí stejný RFCOMM kanál.

Běžný postup vytvoření serveru je rozložen do několika kroků:

1. Získání instance BluetoothServerSocketu

   - provádí se voláním metody listenUsingRfcommWithServiceRecord(String,UUID) třídy BluetoothAdapter.

2. Zahájení naslouchání

   - zavoláním metody accept() získané instance. Tato metoda je blokující, proto by měla být volána ve vlastním vlákně. Při úspěšném spojení klienta je vrácena instance třídy.

3. Ukončení naslouchání

   - po úspěšném přijmutí klienta by měla být volána metoda close(). Na rozdíl od TCP, povoluje RFCOMM připojení pouze jednoho klienta na kanál.

Pro navázání spojení ze strany klienta je zapotřebí nejdříve získat instanci třídy BluetoothDevice. To lze provést užitím třídy BluetoothAdapter. Jednou z možností je provést vyhledávání zařízení. Druhá možnost spočívá v prohledání seznamu již spárovaných zařízení.

Jakmile je získána instance třídy BluetoothDevice lze provést spojení se serverem tímto způsobem:

1. Získání instance BluetoothSocket

   - voláním metody createRfcommSocketToServiceRecord(UUID) získáme instanci třídy BluetoothSocket.

2. Vytvoření spojení se serverem

   - ke spojení se serverem je zapotřebí zavolat metodu connect(). Tímto voláním se provede vyhledání zařízení, kterému odpovídá zadané UUID v předchozím kroku. Tato metoda je blokující a pokud není odpovídající zařízení nalezeno, je po přibližně 12 sekundách vyvolána výjimka.

Po navázání spojení probíhá přenos dat přes instance tříd InputStream a OutputStream. Data jsou během přenosu převedena na pole bajtů.

```
private class ServerThread extends Thread {
    private BluetoothServerSocket mmServerSocket;
    public AcceptThread() {
        mmSSoc = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
     }
     public void run() {
        BluetoothSocket socket = null;
        while (true) {
            try {
                socket = mmSSoc.accept();
            } catch (IOException e) (
            }
            if (socket != null) {
                mmSSoc.close();
                break;
            }
        }
    }
}
```

```
private class ClientThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        mmDevice = device;
        mmSocket = device.createRfcommSocketToServiceRecord(MY_UUID);
    }
    public void run() {
        try {
            mmSocket.connect();
        } catch (IOException connectException) { mmSocket.close(); }
            return;
        }
    }
}
```

Výpis 5: Komunikace prostřednictvím Bluetooth

Teoreticky lze připojit k jednomu zařízení až 7 dalších do sítě piconet. Postup takovéhoto spojení spočívá v naslouchání na různých UUID. Klienti poté zkouší jednotlivá UUID až se jim podaří uskutečnit spojení. V praxi je však tato možnost znesnadněna možnostmi použitých čipů v mobilních zařízeních.

Během vývoje mé aplikace se mi nepodařilo úspěšně spojit více než dvě zařízení. Třetí zařízení bylo ihned po připojení odpojeno.

3.4.5 Síťová komunikace v NDK

S protokoly TCP a UDP lze pracovat také nativně. K tomu je zapotřebí mít nainstalováno Android NDK. Toto API je volitelnou součástí a umožňuje vývoj aplikací v nativním kódu C/C++. Je vhodné zejména pro práci s fyzikou a jinými nízkoúrovňovými procesy, ale lze jej využít také k síťové komunikaci. Slouží k tomu POSIX Socket API, které je součástí NDK.

Na výpisu č.6 je uveden příklad vytvoření TCP soketu v nativním kódu.

```
static int NewTcpSocket(JNIEnv* env, jobject obj)
{
    // Create socket
    LogMessage(env, obj, "Constructing a new TCP socket...");
    int tcpSocket = socket(PF_INET, SOCK_STREAM, 0);
    // Check if socket is properly constructed
    if (-1 == tcpSocket)
    {
        // Throw an exception with error number
        ThrowErrnoException(env, "java/io/IOException", errno);
}
return tcpSocket;
}
```

Výpis 6: Vytvoření soketu v NDK

4 Implementace hry Srdce

4.1 Co jsou srdce?

Srdce jsou karetní hra pro čtyři hráče jejímž cílem je dosáhnou nejmenšího počtu bodů. Hraje se s jedním balíčkem obsahujícím 52 karet. Každý z hráčů dostává 13 karet.

Hráči začnou každé kolo předáním tří karet protihráči (kromě každého čtvrtého kola, kdy se žádné karty nepředávají). Hráč, který má křížovou dvojku, ji vynese a zahájí tak první štych (označení v karetním žargonu pro karty hrané v jednom kole).

Hráči musí ctít barvu. Pokud kartu požadované barvy nemáte, můžete hrát jakoukoli kartou (kromě prvního štychu, kdy nelze vynést srdce nebo pikovou dámu).

Hráč, který vynese nejvyšší kartu, bere štych a začíná další kolo. Ve hře Srdce jsou hodnoty karet seřazeny od esa (nejvyšší) po dvojku (nejnižší).

Hráči mohou začít každý další štych kartou libovolné barvy. Výjimkou jsou srdce. Nelze vynést srdcovou kartu, dokud ji někdo nevynese v předchozím štychu. (Vyjádřeno v karetním žargonu, dokud nebyla srdce prolomena.)

Cílem je předat všechny své srdcové karty spoluhráčům (kteří se vám zase snaží předat svoje). Hra končí, jakmile některý hráč dosáhne 100 bodů. Vyhrává hráč, který má v tomto okamžiku nejnižší celkový počet bodů.

Každá srdcová karta má v závěrečném bodování hodnotu jednoho bodu. Piková dáma je hodnocena 13 body.

4.2 Návrh aplikace

4.2.1 Prostředí

Aplikace byla vytvořena v prostředí Eclipse IDE s nainstalovaným ADT pluginem. Pozadí jednotlivých obrazovek bylo vytvořeno v programu GIMP 2 ve verzi 2.8.6. Obrázky herních karet byly staženy z internetové adresy *https://code.google.com/p/vector-playing-cards/* a následně upraveny v programu Batch Image Resizer 2.88 na potřebnou velikost.

4.2.2 Jádro

Aplikace HotHearts je navržena jako Server—Klient se serverem pracujícím přímo v zařízení. Tento způsob komunikace byl zvolen ze dvou hlavních důvodů:

1. Nezávislost - díky umístění serveru přímo v zařízení není nutná správa externího serveru.

2. Izolovanost - hra muže být spuštěna v režimu Wi-Fi Hotspot. Tím je umožněno připojení klientů i mimo prostředí internetu.

Pro vytvoření jádra aplikace jsem zvolil již vytvořený framework, který byl zveřejněn jako zdrojový kód ke knize *Beginning Android Games* od Marka Zechnera a Roberta Greena [1]. K tomuto frameworku jsem přidal několik balíčků obsahujících třídy reprezentují grafické uživatelské prostředí a logiku samotné hry.

4.2.3 Minimální požadavky

Aby bylo možno aplikaci spustit, je nutné mít operační systém Android ve verzi 2.3 a vyšší.

4.2.4 Popis aplikace

Po spuštění aplikace je uživateli nabídnuto herní menu s možnostmi založit vlastní hru, připojit se ke hře nebo hrát v režimu jednoho hráče.

Pro založení nové hry je nutné stisknout tlačítko *HOST GAME*. Po tomto stisku se zobrazí dialog s výběrem připojení. Na výběr je varianta Wi-Fi a verze Hotspot, ve které se server tváří jako přístupový bod. Po zvolení módu hry je provedena kontrola připojení a následně spuštěna aktivita založení nové hry. Během jejího vytváření je zjištěna IP adresa a vytvořena instance serveru. Server po vytvoření začne naslouchat na zadané adrese a přijímat klienty připojující se do hry. Je-li stisknuto tlačítko *BACK* na herní obrazovce, je server ukončen a uživatel je vrácen do hlavní nabídky. Pokud chce uživatel změnit jméno, klikne do obdélníku se jménem. Tím se vyvolá dialog, ve kterém uživatel vypíše své jméno a potvrdí. Stisknutím tlačítka *START GAME* je na straně serveru vytvořen klient, který se rovněž připojí do hry a následně je spuštěna samotná hra. Podmínkou je dostatečný počet hráčů.

Pokud se hráč rozhodne připojit k již vytvořenému serveru, zvolí tuto možnost z hlavní nabídky. Provede se kontrola, zda je uživatel připojen k bezdrátové síti. Je-li připojen, přepne se aplikace do obrazovky připojení ke hře. V této obrazovce má uživatel na výběr vyhledat servery a následně se k nim připojit. Po stisku tlačítka *FIND HOST*, se aplikace pokusí vyhledat servery na síti, k níž je uživatel připojen. Nalezené servery jsou vypsány pod sebe ve střední části obrazovky. Není-li nalezen žádný server, je to uživateli oznámeno prostřednictvím zprávy na obrazovku. Samotné připojení probíhá stiskem příslušné IP adresy. Po tomto stisku je zobrazena aktivita připojování. Zde uživatel čeká na start samotné hry. Nechce-li uživatel dále čekat, může stiskem tlačítka *BACK* přejít zpět do hlavní obrazovky.

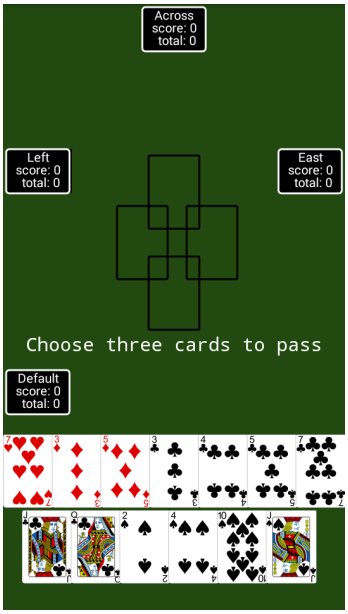
Samotná herní obrazovka je zobrazena na obrázku č.6. Nalézají se zde hráčovy karty, jména všech hráčů, jejich aktuální skóre a celkové skóre.

Další obrázky aplikace jsou uvedeny v příloze A.

4.3 Balíček networking

4.3.1 Třída Utils

V této třídě se nachází obslužné metody, prostřednictvím kterých lze zjišťovat stav sítě, IP adresu zařízení, případně také broadcastovou adresu sítě. Tyto metody fungují na principu prohledávání všech dostupných síťových rozhraní a následném vrácení adresy bezdrátové sítě. Dále tato třída obsahuje metody k vytvoření krátkých informačních zpráv v prostředí systému Android.



Obrázek 6: Herní obrazovka aplikace HOT Hearts

4.3.2 Třída DataPackage

Tato třída slouží jako komunikační prostředek v síti. Obsahuje atributy type a dataString, které ve svém konstruktoru inicializuje. Pomocí atributu type se určuje, o jaký typ zprávy se jedná. Pro účel porovnání jsou zde vytvořeny konstanty, ke kterým je přiřazena hodnota.

4.3.3 Třída Server

Tato třída slouží jako server aplikace. Jejím účelem je přijímat klienty a komunikovat s nimi prostřednictvím protokolů TCP a UDP. Každý klient je zpracováván a obsluhován ve vlastním vlákně. Tato třída obsahuje také instanci třídy GameServer, která spouští samotnou hru. Dále tato třída obsahuje dva atributy ServerSocket pro protokol TCP a jeden třídy DatagramSocket pro protokol UDP. DatagramSocket slouží k příjmu paketů, které mají za účel zjistit přítomnost serveru. Jeden ServerSocket je určen pro komunikaci s klientem, druhý slouží k zasílání herních příkazů a zpráv.

Aby se bylo možno spojit se serverem, je nutné spustit naslouchání na třech portech příslušné IP adresy. To lze provést voláním metody bind(), která přijímá jako parametry čísla těchto portů a IP adresu serveru. V případě neúspěšného startu naslouchání je vyvolána výjimka typu IOException.

K samotnému spuštění serveru dojde po zavolání metody start(). V této metodě jsou spuštěna samostatná vlákna, sloužící k příjmu klientů. Zároveň je nastaven atribut isAccepting, který zajišťuje, že budou tato vlákna korektně ukončena v případě zastavení serveru nebo spuštění hry.

Zjišťování přítomnosti serveru probíhá prostřednictvím protokolu UDP. Server naslouchá na příslušném portu a čeká na příchod paketu. Na základě jeho datové složky rozhoduje, zda se jedná o vyhledávací paket či nikoli. V případě, že se jedná o tento druh paketu, zašle server odpověď. Obsahem odpovědi je počet aktuálně připojených klientů.

```
private Runnable discover = new Runnable()
{
    public void run()
    {
        WifiManager wim = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
        if (wim != null)
        {
            MulticastLock mcLock = wim.createMulticastLock("MLOCK");
            mcLock.acquire();
        }
        MulticastSocket mSocket = null;
        try
        {
            mSocket = new MulticastSocket(discoveryPort);
            while(isAccepting)
            {
                byte[] data = new byte[2];
                DatagramPacket dpacket = new DatagramPacket(data,data.length);
                dpacket.setData(data);
                mSocket.receive(dpacket);
                String message = new String(dpacket.getData()).trim();
                if (message.equals("00"))
                {
                    byte[] ans = intToBytes( list_client_threads .size() );
                    datagramSocket.send(new DatagramPacket(ans,ans.length,dpacket.getAddress(),
                            dpacket.getPort()));
                }
            }
        }
        catch(IOException e){}
        finally
        {
            mSocket.close();
        }
    }
};
```

Výpis 7: Metoda discover()

Jak lze vidět na výpisu č.7 je po zavolání metody discover() vytvořen a nastaven MulticastLock. Následně je inicializována instance třídy MulticastSocket, která přijímá jednotlivé pakety. Po kontrole datové složky paketu následuje odeslání odpovědi, a to přes atribut datagramSocket. Po ukončení vyhledávání je MulticastLock uvolněn a soket uzavřen.

K příjmu klientů jsou určeny metody acceptState a acceptGame.

První metoda volá metodu accept() jedné instance ServerSocketu. Tímto se spustí naslouchání na příslušném portu. Pokud dojde ke spojení ze strany klienta, je vytvořen

objektový proud, který přečte uživatelské jméno klienta. Pokud je toto jméno již použito jiným klientem, je klient odmítnut s žádostí o změnu jména. V opačném případě je vytvořena instance třídy ClientThread, která je uložena do kolekce. Voláním metody start () se spustí nové vlákno, které se stará o samotnou komunikaci s klientem.

```
ClientThread ct = new ClientThread(statusSocket,oos,ois,username);
 list_client_threads .add(ct);
usernames.add(username);
ct.start();
```

Výpis 8: Metoda acceptState()

Ve druhé metodě se rovněž čeká na spojení ze strany klienta. Po navázání spojení je vytvořen soket, který je uložen do kolekce. Tato kolekce je předána instanci třídy GameServer a slouží ke komunikaci v rámci hry.

ClientThread je vnitřní třídou třídy Server a stará se o příjem a odesílání zpráv mezi serverem a klientem. Její run metoda čeká ve smyčce while na objekt typu DataPackage. Pomocí metody getType() je zjištěn druh zprávy a následně je provedena příslušná akce.

```
dp = (DataPackage)ois.readObject();
switch(dp.getType())
{
    case DataPackage.LOGOUT:
    {
        keepListen = false;
        break;
    }
     ...
}
```

Výpis 9: Třída ClientThread

Při odhlášení klienta dojde k přerušení smyčky while a následnému vyvolání metody remove, která dle zadaného identifikátoru, projde kolekce vláken, soketů a uživatelských jmen a odstraní příslušné položky.

```
synchronized void remove(int id)
{
    for(int i = 0; i < list_client_threads .size(); i++)
    {
        ClientThread ct = list_client_threads .get(i);
        if(ct.id == id)
        {
            list_client_threads .remove(i);
            list_game_sockets.remove(i);
            usernames.remove(ct.username);
            return;
        }
    }
}
```

Výpis 10: Metoda remove()

Rovněž je volána metoda close(), která korektně ukončí příslušné proudy.

Ke spuštění hry slouží metoda startGame(). Vyvoláním této metody dojde k nastavení atributu isAccepting na hodnotu false a tím i k ukončení metod discover, acceptState a acceptGame. Od této chvíle již nejsou přijímáni další klienti. Všem klientům je poslána zpráva typu START GAME, která zajistí spuštění herního klienta. Následně je inicializován a spuštěn herní server.

```
public void startGame()
{
    isAccepting = false;
    for(ClientThread ct :  list_client_threads )
    {
        ct.writeMessage(new DataPackage(DataPackage.START_GAME,""));
    }
    game = new GameServer(this,list_game_sockets,usernames);
    game.start();
}
```

Výpis 11: Metoda startGame()

4.3.4 Třída GameServer

Tato třída je potomkem třídy Thread a jejím účelem je spustit instanci samotné hry. Jako parametry jsou jí předány kolekce uživatelských jmen, herních soketů a také instance samotného serveru. Jakmile je ze serveru zavolána metoda start(), je vytvořeno nové vlákno. V tomto vlákně se nejdříve vytvoří jednotliví hráči ve formě kolekce instancí třídy HeartsPlayer. Tato kolekce je předána konstruktoru samotné hry společně s instancí herního serveru. Následně je hra spuštěna.

Po ukončení hry je volána metoda closeGame(). V této metodě se projde kolekcí herních soketů a všem uživatelům je odeslána zpráva o ukončení. K odeslání je použita instance třídy PrintWriter.

4.3.5 Třída Client

Jak z názvu třídy vypovídá, jedná se o třídu, která vytváří a zpracovává komunikaci na klientské straně. Obsahuje metody umožňující vyhledat spuštěné servery a připojit se k nim. Mezi její atributy patří dvě instance třídy Socket.

Metoda, jenž se stará o vyhledání serveru se nazývá findServersOnHotspot. Jejím výstupem je kolekce typu Map obsahující IP adresu serveru spolu s údajem o počtu aktuálně připojených klientů. Metoda vytváří DatagramSocket a zkouší poslat DatagramPacket na adresu a port, zadané jako parametr. Poslední parametr je timeout, tedy doba, po kterou bude soket zkoušet přijímat pakety. Je-li server dostupný, je z adresy vrácen paket, v jehož datové složce je počet připojených klientů. Tento server se poté uloží do mapy. Po uplynutí času zadaného parametrem timeout je vyvolána výjimka typu SocketTimeoutException, při níž je vrácen seznam všech dostupných serverů.

```
public Map<InetAddress,Integer> findServerOnHotspot(InetAddress hostAddress,int port, int
    timeout) throws IOException
{
   Map<InetAddress,Integer> servers = new HashMap<InetAddress,Integer>();
   DatagramSocket dsocket = null;
   try
   {
      dsocket = new DatagramSocket();
      byte[] data = "00".getBytes();
      dsocket.setBroadcast(true);
      dsocket.send(new DatagramPacket(data,data.length,hostAddress,port));
      while(true)
      {
         byte[] ans = new byte[20];
         dsocket.setSoTimeout(timeout);
         DatagramPacket recPack = new DatagramPacket(ans,ans.length);
         dsocket.receive(recPack);

         int clients = bytesToInt(recPack.getData());
         servers.put(recPack.getAddress(),clients);
      }
   }
   catch(SocketTimeoutException stex)
   {
      return servers;
   }
   return null;
}
```

Výpis 12: Metoda findServersOnHotspot()

K samotnému spojení slouží metoda Connect(). V této metodě je nejdříve proveden pokus o připojení k serveru na příslušné IP adrese a portu sloužícímu ke komunikaci. Po úspěšném spojení jsou inicializovány vstupní a výstupní objektové proudy. Důležitým faktorem je pořadí jejich inicializace, neboť by mohlo dojít k tzv. deadlocku. Tato situace nastane v případě, kdy server i klient nejdříve vytvoří vstupní proud. Oba proudy poté čekají na korektní výstupní proud, který obsahuje v hlavičce číslo verze a tzv. magické číslo, dle kterého je proud ověřen.

Dalším krokem je odeslání uživatelského jména a následný příjem odpovědi. Také je spuštěno vlákno sloužící ke komunikaci a proveden pokus o připojení na herním portu. Posledním krokem je inicializace herního klienta.

```
oos.writeObject(username);
message = (String)ois.readObject();
if (message.equals("Welcome")){
   connected = true;
   new ReceiveThread().start();
   gameSocket = new Socket(InetAddress.getByName(serverAddress),gamePort);
   gameClient = new GameClient(this,gameSocket);
}
```

Výpis 13: Metoda Connect

V metodě Connect je po úspěšném navázání spojení spuštěna instance třídy ReceiveThread. Tato instance má za úkol přijímat zprávy ze strany serveru. Tyto zprávy jsou ve formě instancí třídy DataPackage, jejíž atribut type určuje akci, která bude vykonána.

```
dp = (DataPackage)ois.readObject();
switch(dp.getType())
{
    case DataPackage.LOGOUT:
    {
        disconnect();
        break;
    }
    case DataPackage.START_GAME:
    {
        isPlaying = true;
        gameClient.start();
    }
}
catch(IOException e){}
catch (ClassNotFoundException e){}
}
```

Výpis 14: Třída ReceiveThread

4.3.6 Třída GameClient

Tato třída se stará o spojení s herním serverem. Obsahuje atributy a metody potřebné ke spojení se serverem, ale také herní atributy, které slouží ke komunikaci s herní obrazovkou. Síťová část obsahuje objekty typu PrintWriter a BufferedReader, které slouží k příjmu a odesílání zpráv.

Do konstruktoru této třídy je předávána instance klienta a herního soketu, aby bylo možno komunikovat se serverem. Samotná komunikace probíhá v metodě run(), ve které se čeká na příjem paketů ze strany herního serveru. Podle koncového znaku zde dochází k vyvolání určité metody.

4.4 Implementační problémy

Tato aplikace pracuje výhradně v síti Wi-Fi. U spojení prostřednictvím Bluetooth se nepodařilo navázat spojení s více než dvěma zařízeními. Příjem třetího klienta proběhl úspěšně, avšak během několika sekund bylo spojení ukončeno. Ve výpisu z logu v prostředí Android Studio nebyla zjištěna žádná chyba. Tento problém se objevil u všech testovaných zařízení, proto se domnívám, že chyba může být v samotném API. Z tohoto důvodu nebylo toto rozhraní využito ve výsledné aplikaci.

5 Testování aplikace a zátěžové testy síťových rozhraní

Aplikace byla testována na čtyřech telefonech se systémem Android. Byla testována funkčnost samotné hry a také jednotlivé aspekty síťové komunikace. Telefony, které byly využity na testování jsou uvedeny v tabulce č.2

Typ telefonu	Verze OS	Co bylo testováno
ZTE V970	4.1.2	Hra, Server, Klient, Test odezvy, Test propustnosti
Samsung Galaxy ACE	2.3.6	Hra, Klient, Test odezvy, Test propustnosti
Samsung Galaxy Trend	4.0	Hra, Klient
Samsung Galaxy S4	4.4.2	Hra, Server, Klient

Tabulka 2: Mobilní telefony použité pro testování aplikace

Pro účely otestování síťové knihovny byla vytvořena zvláštní aktivita, na které se zkoušely jednotlivé aspekty síťového provozu.

Na aplikaci byly provedeny následující testy:

- Test odezvy - tento test se skládal z opakovaného posílání paketů směrem od serveru ke klientovi a následném měření rychlosti odezvy. Počet opakování byl stanoven na 100. Jako výsledná hodnota byl brán nejnižší naměřený čas odezvy, aby se omezil vliv směrovače na naměřené hodnoty.
- Test propustnosti - na základě tohoto testu byla zjišťována propustnost sítě. Do datové složky paketu byl vložen řetězec o délce milion znaků a odeslán klientovi. Po obdržení odezvy byl zaznamenán čas. Tento test byl proveden 100 krát.

Výsledky testů:

Název testu	Technologie	Třída	Výsledek
Test odezvy	Wi-Fi	Server	46 ms
Test propustnosti	Wi-Fi	Server	158 ms
Test odezvy	Hotspot	Server	37 ms
Test propustnosti	Hotspot	Server	232 ms
Test odezvy	Wi-Fi	GameServer	18 ms
Test propustnosti	Wi-Fi	GameServer	665 ms
Test odezvy	Hotspot	GameServer	18 ms
Test propustnosti	Hotspot	GameServer	806 ms

Z provedených testů vyplynulo, že práce v režimu přístupového bodu má z hlediska síťové komunikace horší výsledky. To může být zapřičiněno nutnou režií samotného připojení. Dále bylo zjištěno, že pro přenos většího množství dat je lepší využít třídy Server, která pracuje na objektovém přístupu k datům.

Testy ukázaly, že mnou navržené rozhraní není nejrychlejší z hlediska přenosu dat. Myslím si však, že u her založených na tazích nebude mít toto vliv na celkový dojem z hraní.

6 Závěr

Mobilní telefon se v dnešní době stal nepostradatelnou součástí života mnohých občanů. U většiny těchto zařízení můžeme nalézt operační systém Android. Aby tomu tak bylo nadále, musí se neustále vyvíjet. Dnes se nachází ve verzi 4.4, také známé pod kódovým označením KitKat. Tato verze podporuje veškeré moderní technologie dnešní doby. Vyvíjet na tuto platformu můžeme ve dvou hlavních vývojových prostředích, Eclipse IDE a Android Studio.

Systém Android nabízí možnost jednoduše komunikovat prostřednictvím protokolu HTTP nebo můžeme zvolit paketovou komunikaci. Výběr protokolu závisí na tom, jak moc chceme mít pod kontrolou samotný přenos. Díky třídě HttpClient a jejím potomkům lze stáhnout obsah webové stránky v několika málo krocích. U paketové komunikace máme na výběr mezi protokoly UDP nebo TCP/IP. Navíc lze po doinstalování Android NDK vytvářet síťové aplikace také v nativním jazyce C++.

Znalosti získané z teoretické části této práce jsem využit k vytvoření knihovny tříd, starající se o vytvoření serveru, klientů a komunikace mezi nimi. Ta probíhá dvěma způsoby. Prvním způsobem je vložení textového řetězce do speciálního objektu, jeho následná serializace a odeslání. Takto v mé knihovně funguje zasílání zpráv typu odhlášení klienta a spuštění hry. Tyto zprávy se netýkají hry jako takové. Druhá možnost spočívá v odeslání textového řetězce přímo. Tímto způsobem komunikuji v samotné hře.

Ukázkovou aplikaci využívající mou knihovnu jsem nazval *HOTHearts*. Jedná se o síťovou variantu hry Srdce určenou pro čtyři hráče. Grafické prostředí a ovládání je vytvořeno s pomocí herního frameworku z knihy Beginning Android Games od Marka Zechnera a Roberta Greena. Aplikace umožňuje založit hru, tedy vytvořit server, který přijímá klienty a řídí hru. Další možností je připojit se k již vytvořené hře jako klient. Jako možné rozšíření současné verze bych viděl přidání komunikačního kanálu mezi uživateli.

Na jednotlivá rozhraní knihovny jsem aplikoval testy, zjišťující rychlost přenosu a propustnost. Výsledky těchto testů ukázaly, že mnou vytvořená knihovna je dostatečně dimenzovaná pro tahové hry. U takovýchto her není kladen důraz na rychlost přenosu. Pro hry komunikující v reálném čase může být doba odezvy limitujícím faktorem.

Některé cíle této práce nebyly splněny. U spojení přes rozhraní Bluetooth jsem narazil na omezení spočívající v možném počtu připojených klientů. Na žádném z mých testovaných telefonů se nepodařilo uskutečnit spojení s více než jedním klientem. Proto nebyl tento způsob přenosu implementován do výsledné aplikace. Rovněž jsem neimplementoval možnost připojení k veřejnému serveru. Důvodem bylo mé soustředění na práci mimo síť internet.

Před touto bakalářskou prací se má tvorba soustředila na vývoj aplikací využívajících pouze data uložená v mobilním telefonu. Díky této práci jsem si rozšířil obzor v oblasti síťových technologií. Rovněž jsem se naučil využít tyto poznatky v praxi.

7 Reference

[1] Zechner, Mario, Green,Robert *Begining Android Games second edition*, Apress, 2012

[2] Google Blog - Where's my Gphone? [online]. 2007 [cit: 2014-03-04]
Dostupné z: http://googleblog.blogspot.cz/2007/11/wheres-my-gphone.html

[3] Android Developers: Welcome. [online]. [cit: 2014-03-04]
Dostupné z: http://developer.android.com/about/index.html

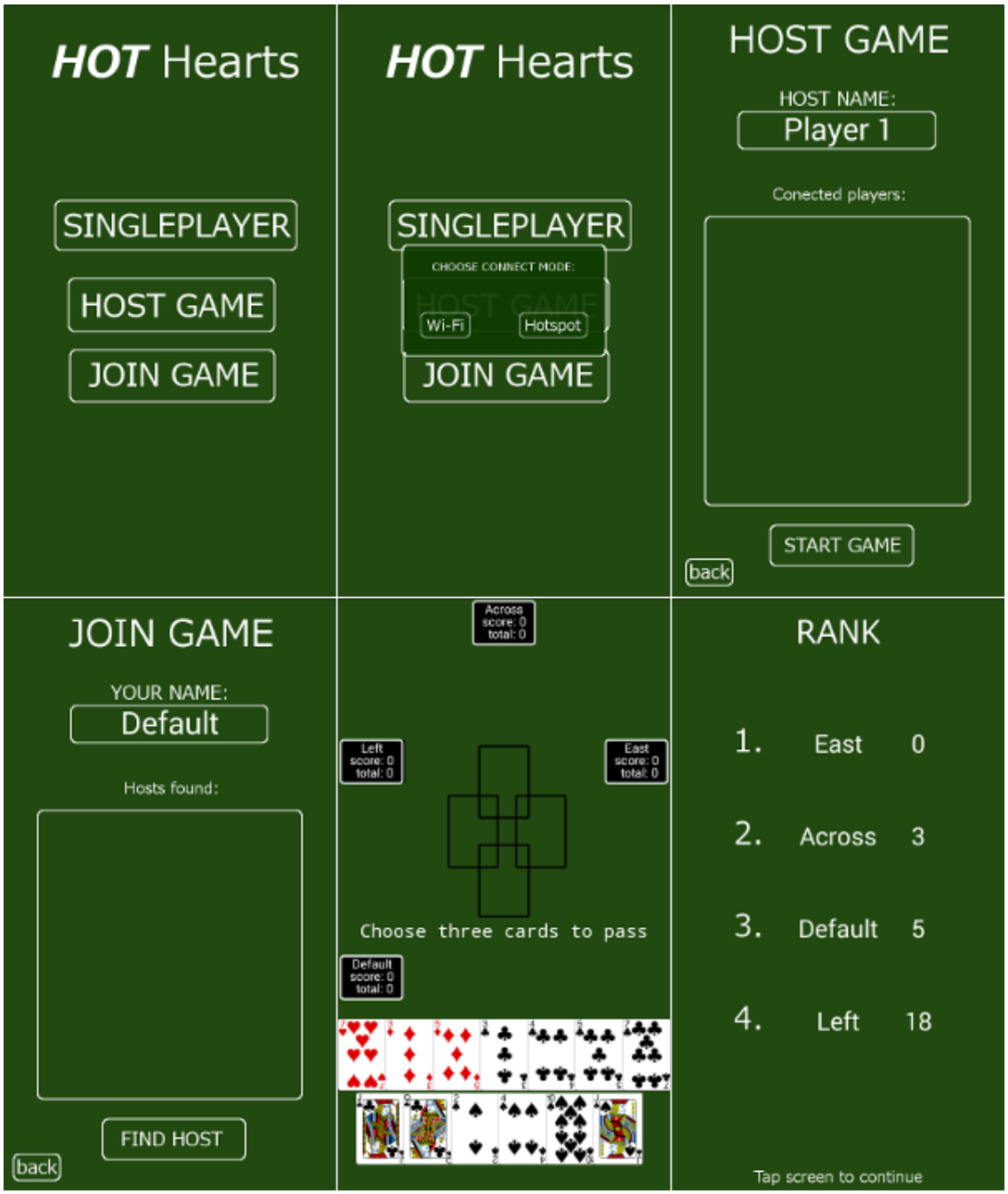
[4] Android Developers: Dashboards. [online]. [cit: 2014-03-04]
Dostupné z : http://developer.android.com/about/dashboards/index.html

[5] Android: A visual history [online]. 2011 [cit: 2014-03-04]
Dostupné z :http://www.theverge.com/2011/12/7/2585779/android-history

[6] Wi-Fi Direct – objevte možnosti nové technologie (vědecké okénko) [online]. 2011 [cit: 2014-03-04]
Dostupné z: http://mobilizujeme.cz/clanky/wi-fi-direct-objevte-moznosti-nove-technologie-vedecke-okenko/

[7] Android Developers: Bluetooth. [online]. [cit: 2014-03-04]
Dostupné z : http://developer.android.com/guide/topics/connectivity/bluetooth.html

A Obrázky ze hry



Obrázek 7: Obrázky ze hry